Computer Architecture

Project 4

Due Date : 2022, 06, 09 23:55 PM

Project 4) Out-of-order 로그 분석 툴 작성

다음 지시사항을 지켜 주시길 바랍니다.

1) project4 여기에 각각 필요한 파일들을 저장해 주세요.
2) tar파일로 압축해 주세요. 압축 파일명은 반드시 project4_학번.tar 가 돼야 합니다.
   예) project4_2015147526.tar
   있어야 하는 파일
   - bitcount_stdout.txt
   - bitcount_debug.txt
   - o3_parser.py
   - result.txt

※ 프로젝트 구현을 위한 소스 코드들을 함께 첨부하였습니다.
ca_proj4_codes/src/cpu/o3/commit_impl.hh
ca_proj4_codes/src/cpu/o3/decode_impl.hh
ca_proj4_codes/src/cpu/o3/fetch_impl.hh
ca_proj4_codes/src/cpu/o3/iew_impl.hh
ca_proj4_codes/src/cpu/o3/SConscript

첨부된 다섯 파일을 Project2에서 사용한 gem5의
src/cpu/o3/ 폴더에 복사해주세요. (모두 덮어씌울 것)

제공된 파일들은 CA_PROJ4 디버그 플래그를 통해 특정 로그를
출력하기 위해 사용됩니다.

본 과제에선 추가로 gem5 코드를 수정할 필요가 없습니다.
해당 파일들을 덮어씌운 뒤, scons build/ARM/gem5.opt
(project 2에서 문제가 발생하셨던 분은 그 때 사용하셨던 커맨드)를
다시 돌려주세요.

Project 4 개요

- gem5의 arm_detailed cpu는 Out-of-order 실행을 지원합니다.
- gem5의 파이프라인은 Fetch - Decode - Issue/Execute/Writeback - Commit 순으로 이루어집니다.
- arm_detailed CPU에서 Execute / Writeback은 out-of-order로 이루어지며, 최종적으로 명령어가 정상적으로 실행됐음을 결정하는 Commit은 In-order로 이루어집니다 (단, 한 번에 Commit 가능한 여러 명령어가 동시에 Commit이 이루어질 수 있습니다).
- arm_detailed에선 Execution이 끝나면 바로 Writeback이 이루어집니다.
- 프로그램의 실행 과정에서 동적으로 실행 된 명령어의 순서를 Sequence Number라 부릅니다. 예를 들어, 반복문에서 I1, I2, I3을 반복적으로 실행하고, I1이 프로그램의 가장 처음 실행된 명령어라면,

명령어 실행	I1	I2	I3	I1	I2	I3	I1	I2	I3
Sequence Number	1	2	3	4	5	6	7	8	9

  위와 같이 진행됩니다. 하나의 정적 (static) 명령어가 동적으로 여러 번 실행될 수 있으므로, Sequence Number를 통해 해당 실행을 구분할 수 있습니다.
- 첨부된 파일들을 덮어씌우신 후, CA_PROJ4를 --debug-flags= 옵션에 추가하는 것으로 명령어 실행에 관한 로그를 출력할 수 있습니다. 해당 로그에서는 각 동적 명령어의
  - Sequence Number
  - 현재 단계 (fetch / decode / issue / execute(start) / writeback / commit 으로 구분)
  - 현재 Tick (시점)
  - 현재 단계가 fetch일 경우 명령어의 어셈블리 코드

```
 77000: system.cpu.fetch: 77000/1/fetch/  mov   fp, #0
 78500: system.cpu.decode: 78500/1/decode
 80000: system.cpu.iew: 80000/1/issue
 80500: system.cpu.iew: 80500/1/execute(start)
 80500: system.cpu.iew: 80500/1/writeback
 81500: system.cpu.commit: 81500/1/commit
127000: system.cpu.fetch: 127000/2/fetch/  mov   lr, #0
127000: system.cpu.fetch: 127000/3/fetch/  ldr   r1, [sp] #4
127000: system.cpu.fetch: 127000/4/fetch/  addi_uop   sp, sp, #4
```

로그 파일 예시

※환경마다 같은 벤치마크여도 Tick 등이 다를 수 있습니다.

  - 이 때, 중간에 버려지는 (Squash) 명령어들은 특정 단계가 나타나지 않을 수 있습니다. 최종적으로 commit이 로그에 존재하지 않는 동적 명령어들은 Squash된 것으로 간주하시면 됩니다.

- 로그 파일을 바탕으로 다음과 같은 결과 파일을 출력하는 Python 프로그램을 작성해 주세요. 이 때, 반드시 commit까지 이루어진 명령어만을 출력해야 합니다.
- 같은 명령어에 대해서 동일한 스테이지에 대한 로그가 여러번 나타날 시, 가장 마지막에 실행된 로그를 바탕으로 표를 작성해주세요.

각 항은 반드시 tab(\t)으로 구분 (반드시 \t하나로만 구분할 것)							
Sequence Number (오름차순)	Operation 종류	Fetch 시점	Decode 시점	Issue 시점	Execute (마지막) 시점	Writeback 시점	Commit 시점

※Operation 종류는 각 명령어의 operation만 출력해 주시면 됩니다. 예를 들어, add r0, r1, r2 명령어에서 operation은 add입니다.

```
300       bne   886000  887500  889000  891500  891500  893000
325       sub   893500  895000  896500  897000  897000  898000
326       cmps  894000  895500  897000  897500  897500  898500
327       ldrls 894000  895500  897000  898500  899500  901000
353       ldr   901500  903000  904500  905500  906500  907500
354       mov   901500  903000  904500  905000  905000  907500
355       mov   901500  903000  904500  907000  907000  908000
356       b     902000  903500  905000  905500  905500  908000
```

결과 예시

※환경마다 같은 벤치마크를 실행해도 Tick 등이 차이가 날 수 있습니다.

Project 4 스펙

- bitcount 벤치마크를 다음과 같은 조건으로 실행한 뒤, stdout-file을 bitcount_stdout.txt로, debug-file을 bitcount_debug.txt로 제출해 주세요.
  - --cpu-type=arm_detailed
  - --caches --l2cache 옵션을 추가할 것
  - 따로 cache size를 커맨드라인에 추가하지 말 것 (gem5 default cache size가 자동으로 사용됩니다)
  - 벤치마크에 입력값을 -o 10 으로 줄 것
  - --debug-flags=CA_PROJ4 를 추가하여 실행할 것
  - --output 조건을 사용하지 말 것 (즉, stdout file에 output이 함께 들어가도록 할 것)
- bitcount_debug.txt를 Project 4 개요의 설명대로 분석할 수 있는 o3_parser.py
  - 반드시 sys.argv[1]로 입력 파일명, sys.argv[2]로 출력 파일명을 받아야 합니다.
- o3_parser.py를 사용하여 bitcount_debug.txt를 분석한 result.txt